UIUCDCS-R-79-965 UILU-ENG 79 1711

Theorem Proving with Abstraction, Part II

by

David A. Plaisted

March 1979

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Theorem Proving with Abstraction, Part II

David A. Plaisted

University of Illinois at Urbana-Champaign

Address: 222 Digital Computer Laboratory
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

This research was supported in part by the National Science Foundation under Grant MCS 77-22830.

Abstract

The concept of an abstraction was defined in Part I of this paper, and some theorem proving strategies based on abstraction were presented. The basic idea is to use the solution of a simple "abstracted" problem as a guide to the solution of a more complicated problem. This idea was formalized to yield a wide class of complete theorem proving strategies for the first-order predicate calculus. In part II, m-abstractions are defined, and their advantages are discussed. They operate on "multiclauses", which are multisets of literals. Several elegant strategies based on m-abstractions are presented. Next, bounded multiclauses are introduced, together with abstractions on them. These have most of the advantages of ordinary multiclauses, but restrict the size of the abstracted search space more. All strategies considered in Part II are complete. Finally, some new classes of abstraction and m-abstraction mappings are presented.

Table of Contents

1. INTRODUCTION

In Part I of this paper, we introduced the concept of an abstraction. We also introduced a complete theorem proving strategy for the first-order predicate calculus. This strategy uses an abstraction to map a given problem onto a simpler problem, and attempts to invert the mapping to obtain solutions to the original problem from solutions to the simplified problem. Abstractions are defined as a class of mappings possessing a few simple mathematically specified properties. We exhibited some sample abstraction mappings, and gave general methods for obtaining many more.

In Part II, we extend the concept of abstraction to "multiclauses", which are multisets of literals. That is, we allow a literal to occur more than once in a multiclause. This added complexity allows much simpler strategies than we could obtain for ordinary clauses. The concept of abstraction is adapted to multiclauses to obtain the "m-abstraction" mappings. These mappings have properties similar to those of abstraction mappings. We present some examples of m-abstraction mappings, and give general methods for obtaining more. One advantage of m-abstraction mappings and multiclauses is that several such mappings can be used together in a natural way in the search for a proof. We also present "bounded multiclauses" and strategies and abstractions related to them. The advantage of bounded multiclauses is that the size of the abstracted search space is smaller than for ordinary multiclauses. In fact, in certain useful special cases, the size of the abstracted search

space is finite. Finally, we present more methods of obtaining m-abstraction mappings (and corresponding mappings for bounded m-clauses).

Part I of this paper, except for sections 2.7 and 2.8 of Part I, is a prerequisite for Part II. We use similar notation as in Part I. In particular, the programming constructs are the same. For loops, we use <u>loop</u>···<u>while</u>···<u>repeat</u> and <u>loop</u>···<u>until</u>···<u>repeat</u>, where the <u>while</u> clause and the <u>until</u> clause can occur at the beginning or at the end of the loop. We also use the fairly standard <u>if</u>···<u>then</u>···<u>else</u>···<u>fi</u> and <u>do</u>···<u>od</u> constructs. For existential formulae, expressions of the form <u>there exists</u> $x_1, x_2, \ldots, x_n$ <u>such that</u> $A(x_1, x_2, \ldots, x_n)$ are used. Here A is a Boolean-valued expression involving the program variables $x_1, x_2, \ldots, x_n$. The construct <u>there exists</u> $x_1, x_2, \ldots, x_n$ <u>such that</u> $A(x_1, x_2, \ldots, x_n)$ has a Boolean value, which is the truth value of the formula $\exists x_1 \exists x_2 \cdots \exists x_n\ A(x_1, x_2, \ldots, x_n)$. Also, if $\exists x_1 \exists x_2 \cdots \exists x_n\ A(x_1, x_2, \ldots, x_n)$ is true, the program variables $x_1, x_2, \ldots, x_n$ are given values making $A(x_1, x_2, \ldots, x_n)$ true. In this way we can specify searches without specifying the details of the search. Thus we can write statements like <u>if there exist</u> $x_1, x_2, \ldots, x_n$ <u>such that</u> $A(x_1, x_2, \ldots, x_n)$ <u>then</u> [do something with $x_1, x_2, \ldots, x_n$] <u>else</u> ··· <u>fi</u>.

2. M-RESOLUTIONS AND M-ABSTRACTIONS

Definition: A multiset M is a set S together with a function g mapping S into the set of positive integers. We refer to S as Set(M) and for $x \in S$, g(x) is denoted by mult(x, M). By convention, mult(x, M) = 0 if $x \notin S$.

Intuitively, a multiset is a set in which elements can occur more than once. For $x \in S$, mult(x, M) tells "how many times" x occurs in M. We write M as $\{n_1{*}x_1, \ldots, n_k{*}x_k\}$ where mult(x, M) = $\sum_{\substack{j \\ x_j=x}} n_j$. Also, 1*x is written as x.

We often regard an ordinary set A as a multiset M in which each element of A occurs exactly once, and in which no other elements occur.

The size |M| of a multiset M = $\{n_1{*}x_1, \ldots, n_k{*}x_k\}$ is defined to be $\sum_{i=1}^{k} n_i$.

Definition: If M1 and M2 are multisets, then their union M1 ⊎ M2 is defined by mult(x, M1 ⊎ M2) = mult(x, M1) + mult(x, M2). Their intersection M1 ⩀ M2 is defined by mult(x, M1 ⩀ M2) = min(mult(x, M1), mult(x, M2)). Their difference M1 - M2 is defined by mult(x, M1 - M2) = min(0, mult(x, M1) - mult(x, M2)). Sometimes we write ⊎ as ∪. Note that Set(M1 ⊎ M2) = Set(M1) ∪ Set(M2).

Definition: If M1 and M2 are multisets, then we write M1 ⊂ M2 (M1 is a sub-multiset of M2) if for all x, mult(x, M1) $\le$ mult(x, M2).

Definition: If M is a multiset and g is a mapping from Set(M) into a set N, then g(M) = $\biguplus_{x \in M}\{g(x)\}$. Thus mult(y, g(M)) = $\sum_{\substack{x \\ f(x)=y}}$ mult(x, M),

and $|g(M)| = |M|$.

Note that for multisets M1 and M2, $g(M1 - M2) = g(M1) - g(M2)$. This is not true of ordinary sets, however.

Definition: A multiclause (or m-clause) is a multiset of literals. That is, with each literal in the clause, a multiplicity is kept, which is a positive integer telling how many times the literal occurs in the multiclause. We can write a multiclause by writing each element the number of times it occurs in the multiclause. Thus {P, P, Q} is a multiclause in which the multiplicity of P is 2 and the multiplicity of Q is 1.

Definition: If C is a multiclause and α is a substitution, then $C\alpha$ is $\{L\alpha : L \in C\}$ where $L\alpha$ is counted the right number of times. That is, $\text{mult}(L1, C\alpha) = \Sigma_{L\varepsilon C,\ L\alpha=L1}\text{mult}(L, C)$. Thus $|C\alpha| = |C|$, and if $C = \{L_1, L_2, \ldots, L_n\}$ then $C\alpha = \{L_1\alpha, L_2\alpha, \ldots, L_n\alpha\}$.

Example: Suppose C is $\{\bar{P}(x), \bar{P}(c), Q(x)\}$. Suppose α is $\{x \leftarrow c\}$. That is, α replaces x by c. Then $C\alpha$ is $\{\bar{P}(c), \bar{P}(c), Q(c)\}$. Note that the literal $\bar{P}(c)$ occurs twice in $C\alpha$.

Definition: Suppose C1 and C2 are multiclauses. Suppose $A1 \subset C1$ and $A2 \subset C2$. (This means that every literal occurs no more times in A1 than in C1, and similarly for A2.) Suppose there exist substitutions $\alpha 1$ and $\alpha 2$ such that for some literal L, $\text{Set}(A1\alpha 1) = \{L\}$ and $\text{Set}(A2\alpha 2) = \{\bar{L}\}$. Let $\alpha 1$ and $\alpha 2$ be most general such substitutions. Then $(C1 - A1)\alpha 1 \uplus (C2 - A2)\alpha 2$ is an m-resolvent of C1 and C2. (Recall the definition of C1 - A1 for multisets C1 and A1, and similarly for C2 - A2.)

Examples: Suppose C1 is $\{\bar{P}(a), \bar{P}(x)\}$ and C2 is $\{P(a)\}$. Then $\{\bar{P}(x)\}$, $\{\bar{P}(a)\}$ and NIL (the empty multiset) are m-resolvents of C1 and C2.

Suppose C1 is $\{\bar{P}, \bar{P}\}$ and C2 is $\{P, Q, Q\}$. Then the following are the m-resolvents of C1 and C2:

$\{\bar{P}, Q, Q\}$

$\{Q, Q\}$.

Ordinary clauses can be viewed as multiclauses in which the multiplicity of each literal in the clause is 1. We have the following results concerning the relation of m-resolution to ordinary resolution.

Theorem 2.1: Suppose C3 is an ordinary resolvent of clauses C1 and C2. Suppose D1 and D2 are m-clauses such that Set(D1) = C1 and Set(D2) = C2. Then there is an m-resolvent D3 of D1 and D2 such that Set(D3) = C3.

Theorem 2.2: Suppose clause C is derivable from set S of clauses by ordinary resolution. Then there is an m-clause D derivable from S by m-resolution such that Set(D) = C. (In the derivation of D, we consider the clauses of S to be m-clauses). Note that if C = NIL then D = NIL also.

Theorem 2.3: Suppose m-clause D3 is an m-resolvent of m-clauses D1 and D2. Then some ordinary resolvent of Set(D1) and Set(D2) subsumes Set(D3).

Theorem 2.4: Suppose S is a set of clauses, and m-clause D is derivable from S by m-resolution. (For this derivation, we consider the clauses of S to be m-clauses.) Then there is an ordinary clause C

derivable from S by ordinary resolution, such that C subsumes Set(D).

Definition: An m-abstraction is a mapping f from multiclauses to (ordinary) sets of multiclauses, satisfying the following properties:

1. If C3 is an m-resolvent of C1 and C2, and D3 ε f(C3), then there exist D1 ε f(C1) and D2 ε f(C2) such that D3 is an m-resolvent of D1 and D2.
2. f(NIL) = {NIL}.

Notice how much simpler the properties of m-abstractions are than those of ordinary abstractions. The following two results, analogous to Theorems 2.1 and 2.2 of Part I for ordinary abstractions, show that m-abstractions are also easy to construct.

Theorem 2.5: Suppose $\emptyset$ is a mapping from literals to literals. Let us extend $\emptyset$ to a mapping from multiclauses to multiclauses as follows:

$$\emptyset(\{n_1{*}L_1, n_2{*}L_2, \ldots, n_k{*}L_k\}) = \{n_1{*}\emptyset(L_1), n_2{*}\emptyset(L_2), \ldots, n_k{*}\emptyset(L_k)\}.$$

Thus $|\emptyset(C)| = |C|$, but $\emptyset$ need not be one-to-one. Suppose $\emptyset$ satisfies the following properties:

1. $\emptyset(\bar{L}) = \overline{\emptyset(L)}$ for all literals L.
2. If the multiclause D is an instance of the multiclause C, then $\emptyset(D)$ is an instance of $\emptyset(C)$.

Then the mapping f defined on multiclauses by $f(C) = \{\emptyset(C)\}$ is an m-abstraction mapping.

Proof: Similar to the proof of theorem 2.1 of Part I.

<u>Theorem 2.6</u>: Suppose F is a set of mappings from literals to literals. For each $\emptyset \in F$, extend $\emptyset$ to a mapping from multiclauses to multiclauses as in the preceding theorem. Suppose that for all $\emptyset \in F$, $\emptyset(\bar{L}) = \overline{\emptyset(L)}$ for all literals L. Suppose also that if multiclause D is an instance of multiclause C, then for all $\emptyset 2 \in F$ there exists $\emptyset_1 \in F$ such that $\emptyset 2(D)$ is an instance of $\emptyset_1(C)$. Define mapping f on multiclauses by $f(C) = \{\emptyset(C):\emptyset \in F\}$. Then f is an m-abstraction mapping. (Note that f(C) is an ordinary set of multiclauses.)

Proof: Similar to the proof of Theorem 2.2 of Part I.

If f is an m-abstraction as in this theorem, then we say f is <u>defined in terms of literal mappings</u>.

<u>Theorem 2.7</u>: If f is an m-abstraction defined in terms of literal mappings, then f also satisfies the following property:

For all multiclauses C1 and C2, if C1 subsumes C2, then for all $D2 \in f(C2)$ there exists $D1 \in f(C1)$ such that D1 subsumes D2.

(For multiclauses, we say D1 subsumes D2 if there exists a substitution θ such that $D1\theta \subset D2$, that is, for all $L \in D1\theta$, $mult(L, D1\theta) \leq mult(L, D2)$.) This result will be useful later on in obtaining strategies to test if an m-clause is a logical consequence of a set of m-clauses.

2.1 *Examples of M-Abstractions*

These m-abstractions are obtained from the abstractions presented in Section 2.1 of Part I, by counting each literal the right number of times.

1. *The ground m-abstraction.*

If C is the m-clause $\{L_1, L_2, \ldots, L_k\}$ then $f(C) = \{\{L_1\theta, L_2\theta, \ldots, L_k\theta\}: L_i\theta$ are ground literals, for $1 \le i \le k\}$.

2. *The propositional m-abstraction.*

If C is the m-clause $\{L_1, L_2, \ldots, L_k\}$ then f(C) is {D} where D is the m-clause $\{L_1', L_2', \ldots, L_k'\}$, and L_i' is obtained from L_i by deleting all arguments of the predicate symbol. Thus if L_i is $P(t_1, \ldots, t_n)$ then L_i' is P and if L_i is $\bar{P}(t_1, \ldots, t_n)$ then L_i' is $\bar{P}$.

Similarly, we can define m-abstractions based on renaming of predicate or function symbols, changing the signs of literals, changing the order of the arguments of various function or predicate symbols, or removing arguments from various function or predicate symbols. Note as before that the renaming of function and predicate symbols need not be 1 - 1. That is, the names of two distinct predicate symbols can be made the same, and similarly for function symbols.

3. *A semantic m-abstraction.*

Suppose I is an interpretation of the set of clauses over some set of function and predicate symbols. Let $\mathcal{D}$ be the domain of the interpretation I. With each ground literal L we associate a literal L' as in the definition of semantic abstraction from Part I. With a ground m-clause $C = \{L_1, \ldots, L_k\}$ we associate the m-clause $C' = \{L_1', \ldots, L_k'\}$ where L_i' is associated with L_i as indicated above. Note that $|C'| = |C|$. If C1 is an arbitrary m-clause, then $f(C1) = \{D{:}D$ is associated with C for some ground instance C of C1$\}$. Thus if I is the usual interpretation of the integers, then $\{3 \le 3, 3 \le 3\}$

is an m-abstraction of $\{x \leq y, y \leq x\}$. We call f the m-abstraction obtained from I. As before, semantic m-abstractions seem to be particularly useful when the domain $\mathcal{D}$ is finite, because then f(C) is a finite set of multiclauses for all C.

We can define the composition $f_1 f_2$ of m-abstractions f_1 and f_2, and show as before that it is an m-abstraction if f_1 and f_2 are. Also, the union of two m-abstractions is an m-abstraction. Moreover, it is easy to show that if f_1 and f_2 are m-abstractions defined in terms of literal mappings, then the union and composition of f_1 and f_2 are also defined in terms of literal mappings. Inverses of m-abstractions can be defined (if they exist). An m-abstraction which has an inverse really hasn't thrown away any information. Perhaps it should be called an m-isomorphism.

2.2 *M-Abstractions of M-Resolution Proofs*

We now indicate how m-abstractions can be used to guide the search for a proof of an m-clause C from a set S of m-clauses. We omit some details because the development is analogous to that for ordinary abstractions. The concepts of an m-abstraction proof, the depth of an m-abstraction proof, et cetera are defined in a way analogous to the way these concepts were defined for ordinary resolution. We denote the nodes of an m-resolution proof V by Nodes(V) as before, and denote the m-resolutions of V by MRes(V). An m-resolution of V is a triple $\langle N1, N2, N3 \rangle$ of nodes of V such that label(N3) is an m-resolvent of the m-clauses label(N1) and label(N2). We require that if

$\langle N1, N2, N3\rangle \in$ MRes(V) then $\langle N2, N1, N3\rangle \in$ MRes(V), as before. Also, if M1 and M2 are two relations between multiclauses and V1 and V2 are two m-resolution proofs, we define (M1; M2)(V1, V2) as before. We abbreviate (M; M)(V1, V2) as M(V1, V2). If f is an m-abstraction mapping and S is a set of m-clauses, we write f(S) to denote the set $\bigcup_{C \in S} f(C)$. Note that f(S) is an ordinary set of multiclauses.

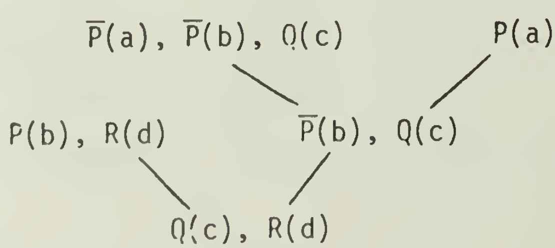
<u>Theorem 2.8</u>: Suppose V2 is an m-resolution proof of m-clause C' from set S of multiclauses. Suppose f is an m-abstraction mapping, and D' $\in$ f(C'). Let M(D, C) be the relation "D $\in$ f(C)". Then there is an m-resolution proof V1 of D' from f(S) such that M(V1, V2) is true.

This is a much better result than for ordinary abstractions. Also, the depth of V1 is the same as the depth of V2.

This result relates to ordinary resolution in the following way: Suppose clause C' is derivable from set S of clauses by ordinary resolution. Suppose f is an m-abstraction mapping. Then there exists an m-clause C1 such that Set(C1) = C', and such that for all D $\in$ f(C1), D is derivable from f(S) by m-resolution. Later we discuss how m-abstractions can help to find ordinary resolution proofs of clauses other than NIL.

Examples

1. Consider the following m-resolution proof: (This is also an ordinary resolution proof)



This example is the same as Example 2 of Section 2.4 of Part I. If we use the propositional m-abstraction, we obtain the following m-abstracted proof:

$$
\begin{array}{ccccccc}
 & & & \bar{P},\bar{P},Q & & P & \\
 & & & \diagdown & & \diagup & \\
 & P,R & & & \bar{P},Q & & \\
 & \diagdown & & \diagup & & & \\
 & & Q,R & & & &
\end{array}
$$

2. Consider the following m-resolution proof, taken from Example 3 of Section 2.4 of Part I: (This is also an ordinary resolution proof.)

$$
\begin{array}{ccccccc}
 & & & \bar{P}(a),\ \bar{P}(b),\ Q(c) & & P(a) & \\
 & & & \diagdown & & \diagup & \\
 & \bar{Q}(c),\ Q(b) & & & \bar{P}(b),\ Q(c) & & \\
 & \diagdown & & \diagup & & & \\
 & & \bar{P}(b),\ Q(b) & & & &
\end{array}
$$

Using the propositional m-abstraction, we obtain the following m-abstracted m-resolution proof:

$$
\begin{array}{ccccccc}
 & & & \bar{P},\bar{P},Q & & P & \\
 & & & \diagdown & & \diagup & \\
 & \bar{Q},Q & & & \bar{P},Q & & \\
 & \diagdown & & \diagup & & & \\
 & & \bar{P},Q & & & &
\end{array}
$$

3. A COMPLETE STRATEGY FOR A SINGLE M-ABSTRACTION

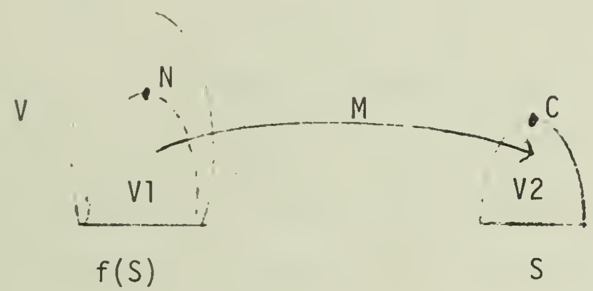
We define the procedure "ndfindm" analogous to "ndfind" of Part I but for multiclauses and m-resolution. This procedure uses m-abstracted proofs as a guide in the search for an m-resolution proof. Suppose f is an m-abstraction mapping. Let M(D, C) be the relation "D ε f(C)". We are given an m-resolution proof V from f(S), and want to find all proofs V2 from S such that M(V, V2) is true. With each node N of V, we keep a set m-clauses(N) of m-clauses derived from S by m-resolution.

```
procedure ndfindm(V, S, f);
   ⟦assume that m-clauses(N) = {C ε S: label(N) ε f(C)}
    for all initial nodes N of V, and m-clauses(N) = ∅
    for all non-initial nodes N of V⟧

   loop

      while (there exist nodes N1, N2, N of V and m-clauses
             C1, C2, C such that

             1.  ⟨N1, N2, N⟩ ε MRes(V)

             2.  C1 ε m-clauses(N1) and C2 ε m-clauses(N2)

             3.  C is an m-resolvent of C1 and C2

             4.  C ∉ m-clauses(N)

             5.  label(N) ε f(C));

      add C to m-clauses(N);

   repeat

end ndfindm;
```

It is not difficult to show that when "ndfindm" exists, then for all nodes N in V, m-clauses(N) will contain exactly the m-clauses C having the following property:

There exists a minimal proof V1 from f(S) such that V1 is an initial sub-proof of V, and such that N is the unique terminal node of V1, and there exists an m-resolution proof V2 from S such that M(V1, V2) is true, and such that C = Result(V2).

See figure 1. Recall that M(D, C) is the relation "D ε f(C)". In addition, if W is any m-resolution proof from S such that M(V1, W) is true for some initial sub-proof V1 of V, then W is isomorphic to an initial sub-proof of the m-resolution proof generated by "ndfindm". We could also define a depth-first search procedure analogous to the procedure "findclauses" of Part I.



Clauses generated by "ndfindm"

Figure 1

In order to analyze the procedure "ndfindm" and related procedures, we introduce the concept of the m-abstraction of an m-resolution proof.

<u>Definition</u>: Suppose S is a set of multiclauses and T is an m-resolution proof from S. Suppose f is an m-abstraction mapping. Let M(D, C) be the relation "D ε f(C)", as before. Then we say T $\underset{f}{\rightarrow}$ U

if U is an m-resolution proof from S such that M(U, T) is true. In Theorem 2.8 we stated that some such U exists. Possibly more than one such proof U will exist. Note that if $T \xrightarrow[f]{} U$ then the depth of U is the same as the depth of T. We have the following result:

Theorem 3.1: Suppose T is a minimal m-resolution proof of an m-clause C' from a set S of m-clauses. Suppose f is an m-abstraction mapping. Then for every m-clause D' in f(C'), there exists an m-resolution proof U from f(S) such that $T \xrightarrow[f]{} U$ and such that Result(U) is defined and equal to D'.

Suppose T is an m-resolution proof of m-clause C' from set S of m-clauses. Suppose U is an m-resolution proof from f(S) such that $T \xrightarrow[f]{} U$. Suppose U is an initial sub-proof of V. Then if ndfindm(V, S, f) is called, it will actually generate a proof of C'. (In fact, it will generate a proof isomorphic to T.) Note that this is an improvement over the situation for ordinary abstractions. A similar result is true for the procedure "findclauses", adapted to m-abstractions and m-resolutions. Also, if T has depth d then U will have depth d. Therefore, if we want to see if a proof of C' exists at depth d, we can choose some m-clause D' in f(C') and need only call ndfindm(V, S, f), where V contains all proofs U from f(S) such that U is of depth d and such that D' is the unique terminal clause of U.

This idea is the basis of the following complete theorem proving strategy based on m-abstractions. The generation of V and V1 in this procedure is similar to the generation of V and V1 in "proofsearch1" of Part I, and notation is similar. In particular, nodes of V and V1 are

of the form $\langle D1, d_1 \rangle$, where D1 is an m-abstraction of some m-clause and d_1 is an integer giving the depth of the node. If $N = \langle D1, d_1 \rangle$ then we write label(N) = D1 and depth(N) = d_1. The proof V1 is constructed so that if U is any minimal m-resolution proof of D' from f(S) such that U has depth d, then U is isomorphic to an initial sub-proof of V1.

<u>procedure</u> proofsearch2(S, C', f);

⟦attempt to construct a proof of C' from S using m-abstraction mapping f. This is a complete theorem proving strategy.⟧

choose D' in f(C');

S1 $\leftarrow \{\langle D, 0\rangle\ (\exists C \in S) D \in f(C)\}$;

for all $\langle D, 0\rangle \in$ S1 <u>do</u>

m-clauses($\langle D, 0\rangle$) $\leftarrow \{C \in S\text{: } D \in f(C)\}$ <u>od</u>;

<u>for</u> d = 1 <u>to</u> ∞ <u>until</u> C' is generated from S <u>do</u>

⟦look for a proof of C' from S of depth d⟧

let V be the smallest m-resolution proof such that

a) S1 $\subset$ Nodes(V)

b) If $\langle B1, d_1\rangle \in$ Nodes(V) and $\langle B2, d_2\rangle \in$ Nodes(V) and $d_1 < d$ and $d_2 < d$ and B3 is an m-resolvent of B1 and B2 then $\langle B3, d_3\rangle \in$ Nodes(V) and $\langle\langle B1, d_1\rangle, \langle B2, d_2\rangle, \langle B3, d_3\rangle\rangle \in$ MRes(V) where $d_3 = 1 + \max(d_1, d_2)$;

let V1 be the smallest sub-proof of V such that

a) If $\langle D', d\rangle \in$ Nodes(V) then $\langle D', d\rangle \in$ Nodes(V1)

b) If $\langle N1, N2, N3\rangle \in$ MRes(V) and N3 $\in$ Nodes(V1) then N1 $\in$ Nodes(V1) and N2 $\in$ Nodes(V1) and $\langle N1, N2, N3\rangle \in$ MRes(V1);

⟦note: V can be found by exhaustive search and V1 can be obtained by deleting nodes and m-resolutions from V. Possibly V1 can be obtained by applying more levels of m-abstraction, also.⟧

for all new non-initial nodes N of V1 <u>do</u> m-clauses(N) $\leftarrow \emptyset$ <u>od</u>;

⟦all initial nodes of V1 will be in S1 and so will have m-clauses assigned⟧

ndfindm(V, S, f)

<u>od</u>;

end proofsearch2;

Since m-abstracted m-clauses will usually be simpler than the original m-clauses, we would expect the construction of V to be easier than an exhaustive search for a proof of C from S. If the search for a proof of D' from f(S) is too complicated to do directly, another m-abstraction mapping can be applied to f(S) to direct the search for a proof of D'. Extending this idea, we can see that any number of "levels of m-abstraction" can be used together in the search for a proof.

As before, we cannot do tautology deletion or subsumption deletion on m-clauses generated from f(S) by m-resolution. The only allowable deletion strategy is to delete variants of an m-clause that has already been derived. However, as with "proofsearch1", <u>any</u> complete theorem proving strategy can be used in the original space, as long as the abstracted space is generated exhaustively. For example, we can restrict "ndfindm" in proofsearch2 so that it only generates resolvents from S according to some complete m-resolution strategy. Thus the search from S would be restricted <u>both</u> by the m-resolution strategy <u>and</u> by the abstracted search space. One promising strategy would be locking resolution, adapted to m-clauses in the appropriate way. It is also possible to restrict the abstracted search space according to some complete strategy, if the m-abstraction satisfies certain properties. For example, if the m-abstraction is defined in terms of literal mappings and preserves signs of literals, then the m-abstraction of a P1-deduction [1] will always be a P1-deduction, and so we can use P1-deduction in both the original space and in the abstracted space. Also, if the indices of the

literals are assigned properly, we can do locking resolution in both the original space and in the abstracted space, for m-abstractions defined in terms of literal mappings. Finally, if the m-abstraction is defined in terms of literal mappings $\emptyset$ such that the predicate symbol of $\emptyset(L)$ is the same as the predicate symbol of L, then we can do m-resolution with ordering of predicate symbols in the original space and in the abstracted space. That is, the predicate symbols are linearly ordered, and in each m-resolution, the literals resolved away must have predicate symbols that are maximal in each clause in the ordering. If the m-abstraction is defined in terms of literal mappings that preserve both signs and predicate symbols of literals, then various combinations of hyper-resolution [1] and ordering can be done in both the abstracted space and in the original space. The improvement to be gained by the use of such complete strategies is probably small compared to the improvement to be gained by the use of m-abstractions. However, even if the strategies only help by a factor of 2 or 3, that will be significant.

Notice that the search space will tend to get smaller as the depth of inference approaches the maximum depth d. This is because the abstracted m-clauses near depth d will be restricted to m-clauses from which D' can be derived in a small number of steps. Thus the m-clauses derived from S at depths near d will also be restricted. This restriction of the search space near the maximum depth contrasts greatly with the behavior of most uniform proof procedures, for which the size of the search space gets larger and larger as the depth of inference increases. At

intermediate depths, the size of the search space will probably be the largest. The strategies "proofsearch3" and "proofsearch4", to be presented later, also restrict the search space as the depth of inference approaches its maximum value, as does the strategy "proofsearch1" presented in Part I.

Another property of strategies based on abstraction is that they automatically choose which m-clauses of S appear to be "relevant" to the problem at hand. That is, an m-clause C of S will not even be used at all unless some abstraction D of C appears in a depth d proof of D' from f(S). Therefore, strategies based on abstraction may be useful when there are many input clauses.

We now discuss methods of using more than one m-abstraction at the same time in the search for a proof.

4. USING MORE THAN ONE M-ABSTRACTION AT THE SAME TIME

Definition: Suppose M is a predicate on k-tuples of m-clauses. That is, if $C_1, C_2, \ldots, C_k$ are m-clauses, then either $M(C_1, C_2, \ldots, C_k)$ is true or $M(C_1, C_2, \ldots, C_k)$ is false. We extend M to a predicate on k-tuples of m-resolution proofs in the following way: Suppose $U_1, U_2, \ldots, U_k$ are m-resolution proofs. Then $M(U_1, U_2, \ldots, U_k)$ is true iff all U_i are of the same shape and there exist shape correspondences $\underset{i}{\sim}$ between U_i and U_{i+1} such that for all nodes N_1 in $U_1, \ldots, N_k$ in U_k, if $N_1 \underset{1}{\sim} N_2$ and $N_2 \underset{2}{\sim} N_3$ and ... and $N_{k-1} \underset{k-1}{\sim} N_k$ then $M(\text{label}(N_1), \text{label}(N_2), \ldots, \text{label}(N_k))$ is true. Note that if M_1 and M_2 are relations such that $M_1(C_1, \ldots, C_k) \supset M_2(C_1, \ldots, C_k)$ for all m-clauses $C_1, \ldots, C_k$, then $M_1(U_1, \ldots, U_k) \supset M_2(U_1, \ldots, U_k)$ for all m-resolution proofs $U_1, \ldots, U_k$.

Let f be an m-abstraction mapping, and let M(D, C) be the relation "$D \varepsilon f(C)$" on m-clauses. We know that if $T \underset{f}{\rightarrow} U$ then M(U, T) is true. Suppose $f_1, f_2, \ldots, f_k$ are m-abstraction mappings. Let $U_1, U_2, \ldots, U_k$ be m-abstraction proofs such that $T \underset{f_i}{\rightarrow} U_i$ for $1 \leq i \leq k$. Let $M'(D_1, D_2, \ldots, D_k)$ be the relation $(\exists C)[D_1 \varepsilon f_1(C)$ and ... and $D_k \varepsilon f_k(C)]$. Then it is not difficult to show that $M'(U_1, U_2, \ldots, U_k)$ is true.

This suggests a search strategy. Suppose we are looking for a proof of m-clause C' from set S of m-clauses. Choose $D_i' \varepsilon f_i(C)$ for $1 \leq i \leq k$. Find proofs U_i of D_i' from $f_i(S)$ such that $M'(U_1, U_2, \ldots, U_k)$ is true. It seems unlikely that such proofs U_i would exist unless a corresponding proof of C' from S exists. If such proofs U_i are found, use them to guide the search for a proof T of C' from S.

The relation M' may be too difficult to compute. However, there will often be other relations M_1' that are easy to compute, such that $M'(D_1, D_2, \ldots, D_k)$ implies $M_1'(D_1, D_2, \ldots, D_k)$. We can then look for proofs $U_1, \ldots, U_k$ such that $M_1'(U_1, \ldots, U_k)$ is true. For example, $M_1'(D_1, \ldots, D_k)$ can specify that $|D_1| = |D_2| = \ldots = |D_k|$. This will work if all the f_i are defined in terms of literal mappings. If the f_i preserve numbers of various predicate symbols for $1 \leq i \leq k$, then $M_1'(D_1, D_2, \ldots, D_k)$ can specify that all D_i have the same number of literals with various predicate symbols. If the f_i preserve <u>signs</u> and predicate symbols of literals, this can also be reflected in M_1'. If $T \xrightarrow[f_i]{} U_i$ for $1 \leq i \leq k$, then $M_1'(U_1, \ldots, U_k)$ will be true. Therefore we can look for proofs U_i of D_i' from $f_i(S)$ such that $M_1'(U_1, \ldots, U_k)$ is true, and use these proofs to guide the search for a proof of C' from S. This will still yield a complete theorem proving strategy. Note that if $T \xrightarrow[f_i]{} U_i$ then the depth of the proof T is the same as the depth of U_i for all i, $1 \leq i \leq k$. This fact can be used to search for proofs T in order of increasing depth. As before, if the search for proofs D_i' from $f_i(S)$ is still too hard, more levels of abstraction can be used.

4.1 *A Strategy Without Matching*

The following procedure uses more than one m-abstraction at the same time in the search for a proof. However, it does not match the m-abstracted proofs up with each other beforehand. That is, we do not test whether all the m-abstracted proofs have the same shape, and

satisfy the appropriate relation on proofs. This saves the effort needed to do the matching, but probably increases the size of the search space. Even so, this procedure should have a smaller search space than "proofsearch2", which only uses one m-abstraction.

procedure proofsearch3(S, C', $\{f_1, f_2, \ldots, f_n\}$);

⟦search for a proof of C' from S using the set $\{f_1, \ldots, f_n\}$ of (not necessarily distinct) m-abstractions. This is a complete strategy.⟧

<u>for</u> i = 1 <u>to</u> n <u>do</u> choose D_i' in $f_i(C')$ <u>od</u>;

<u>for</u> i = 1 <u>to</u> n <u>do</u> $S_i \leftarrow \{\langle D, 0\rangle : (\exists C \in S)\ D \in f_i(C)\}$ <u>od</u>;

<u>for</u> d = 1 <u>to</u> ∞ <u>until</u> C' is derived from S <u>do</u>

⟦look for a proof of depth d⟧

<u>for</u> i = 1 <u>to</u> n <u>do</u>

$V_1^i \leftarrow$ the smallest m-resolution proof such that

a) $S_i \subset \text{Nodes}(V_1^i)$ and

b) if $\langle D1, d_1\rangle \in \text{Nodes}(V_1^i)$ and $\langle D2, d_2\rangle \in \text{Nodes}(V_1^i)$ and $d_1 < d$ and $d_2 < d$ and D3 is an m-resolvent of D1 and D2, then $\langle D3, d_3\rangle \in \text{Nodes}(V_1^i)$ and $\langle\langle D1, d_1\rangle, \langle D2, d_2\rangle, \langle D3, d_3\rangle\rangle \in \text{MRes}(V_i)$ where $d_3 = 1 + \max(d_1, d_2)$;

$V_2^i \leftarrow$ the smallest m-resolution proof such that

a) if $\langle D_i', d\rangle \in \text{Nodes}(V_1^i)$ then $\langle D_i', d\rangle \in \text{Nodes}(V_2^i)$ and

b) if $N3 \in \text{Nodes}(V_2^i)$ and $\langle N1, N2, N3\rangle \in \text{MRes}(V_1^i)$ then $N1 \in \text{Nodes}(V_2^i)$ and $N2 \in \text{Nodes}(V_2^i)$ and $\langle N1, N2, N3\rangle \in \text{MRes}(V_2^i)$

<u>od</u>;

⟦V_2^i represents the depth d proofs of D_i' from $f_i(S)$. These can be found by exhaustive search as indicated above, or by using more levels of m-abstraction.⟧

$W \leftarrow S$;

⟦If node N is of the form ⟨D1, d_1⟩ then we say label(N) = D1 and depth(N) = d_1.⟧

<u>loop</u>

<u>while</u>(C' $\notin$ W <u>and</u> <u>there</u> <u>exist</u> m-clauses C1, C2, and C3 <u>such</u> <u>that</u>

a) C1 ε W and C2 ε W and C3 $\notin$ W

b) C3 is an m-resolvent of C1 and C2

c) for all i, $1 \leq i \leq n$, there exists ⟨N1, N2, N3⟩ ε MRes($V_2{}^i$) such that depth(C1) = depth(N1), depth(C2) = depth(N2), label(N1) ε f_i(C1), label(N2) ε f_i(C2), and label(N3) ε f_i(C3));

add C3 to W;

⟦It may help to choose C1, C2, and C3 so that max(depth(C1), depth(C2)) is as large as possible.⟧

<u>repeat</u>;

<u>od</u>;

<u>end</u> proofsearch3;

4.2 A *Strategy with Matching*

We now present another, similar strategy which uses more than one m-abstraction at the same time. This strategy attempts to "match up" the m-abstracted proofs before looking for a proof from the original set of m-clauses.

As before, if M1 and M2 are relations on k-tuples of m-clauses and $T_1, T_2, \ldots, T_k$ are m-resolution proofs, we define (M1; M2)($T_1, T_2, \ldots, T_k$). Here M1 specifies the relation on clauses at initial nodes and M2 specifies the relation on clauses at non-initial nodes. Suppose $S_1, S_2, \ldots, S_k$ are sets of m-clauses. Suppose that V_i is a set of m-resolutions from S_i. The procedure "match" finds all k-tuples ⟨$T_1, T_2, \ldots, T_k$⟩ of m-resolution proofs from $S_1, S_2, \ldots, S_k$, respectively, such that T_i is an initial sub-proof of V_i for $1 \leq i \leq k$ and such that

$(M1; M2)(T_1, T_2, \ldots, T_k)$ is true. These k-tuples are not generated explicitly, but implicitly, in a manner to be described.

Definition: A vector m-resolution is a triple $\langle u, v, w\rangle$ of nodes such that for some k, the labels of u, v, and w are k-tuples of m-clauses. We require that for all i, $1 \leq i \leq k$, the i^{th} component of label(w) must be an m-resolvent of the i^{th} component of label(u) and the i^{th} component of label(v).

Definition: A vector m-clause is a k-tuple of m-clauses, for some k.

We define vector m-resolution proofs in the same way that m-resolution proofs and ordinary resolution proofs were defined. If V is a vector m-resolution proof, then we require that all the vector m-clauses in V must have the same number of components. We write VMRes(V) to indicate the set of vector m-resolutions of a vector m-resolution proof V.

Suppose D is a k-tuple of m-clauses. We write D_i to refer to the i^{th} component of $\bar{D}$, and we write $\bar{D}$ as $\langle D_1, D_2, \ldots, D_k\rangle$. We use similar notation for k-tuples $\bar{x}$ of nodes.

Definition: Let $\pi_i(\bar{D}, C)$ be the relation specifying that D_i is C. Here $\bar{D}$ is a vector of m-clauses and C is an m-clause.

Definition: If V is a vector m-resolution proof and T is an ordinary m-resolution proof, then we write $\pi_i(V, T)$ if there is a shape correspondence $\sim$ between V and T such that if $N1 \in$ Nodes(V) and $N2 \in$ Nodes(T) and $N1 \sim N2$ then π_i(label(N1), label(N2)) is true. Thus if $\pi_i(V, T)$ is true, T is the "i^{th} component" of the proof V. We are using the usual definition of a relation between proofs here.

The procedure "match($V_1, V_2, \ldots, V_k$, V, M1, M2)" outputs a vector m-resolution proof V having the following property:

Assume $V_1, V_2, \ldots, V_k$ are m-resolution proofs. Suppose $T_1, T_2, \ldots, T_k$ are initial sub-proofs of $V_1, V_2, \ldots, V_k$, respectively. Also, suppose $(M1; M2)(T_1, T_2, \ldots, T_k)$ is true. Then there is an

initial sub-proof W of V such that $\Pi_i(W, T_i)$ is true, for $1 \leq i \leq k$. Thus W has T_i as its "i^{th} component", and W represents the matching up of the proofs $T_1, T_2, \ldots, T_k$. The proof V therefore represents all possible ways of matching up initial sub-proofs of $V_1, V_2, \ldots, V_k$.

<u>procedure</u> match($V_1, V_2, \ldots, V_k$, V, M1, M2);

⟦returns with V a vector m-resolution proof such that for all $T_1, T_2, \ldots, T_k$, if T_i is an initial subproof of V_i, $1 \leq i \leq k$, and if (M1; M2)($T_1, T_2, \ldots, T_k$) is true, then there is an initial sub-proof W of V such that $\Pi_i(W, T_i)$ is true for $1 \leq i \leq k$.⟧

VMRes(V) ← ∅;

Nodes(V) ← {⟨N1, N2, ..., Nk⟩: Ni is an initial node in V_i and

M1(label(N1), label(N2), ..., label(Nk)) is true};

⟦for a node N of V of the form ⟨N1, N2, ..., Nk⟩ we say

label(N) = ⟨label(N1), label(N2), ..., label(Nk)⟩⟧

<u>loop</u>

<u>while</u>(<u>there exist</u> nodes $\bar{x}$ and $\bar{y}$ of V <u>such that</u> $\bar{x}$ and $\bar{y}$ have not

been resolved together yet);

⟦resolve $\bar{x}$ and $\bar{y}$⟧

for all $\bar{z}$ such that $\langle x_i, y_i, z_i \rangle \in$ MRes(V_i) for all i, $1 \leq i \leq k$,

and such that M2(label(z_1), label(z_2), ..., label(z_k)), <u>do</u>

add $\bar{z}$ to Nodes(V);

add $\langle \bar{x}, \bar{y}, \bar{z} \rangle$ and $\langle \bar{y}, \bar{x}, \bar{z} \rangle$ to VMRes(V);

<u>od</u>;

<u>repeat</u>;

<u>end</u> match;

The use of a "divide and conquer" approach may increase the efficiency of the matching procedure. For example, we may match $V_1, V_2, \ldots, V_{\lfloor k/2 \rfloor}$ up first, then match $V_{\lfloor k/2 \rfloor + 1}, V_{\lfloor k/2 \rfloor + 2}, \ldots, V_k$ up, and finally match $V_1, V_2, \ldots, V_k$. This would help because in the final matching phase we would not even need to consider vector m-clauses that had been eliminated in earlier matching phases.

In practice, we may want to economize on the representation of V. For example, if $Q_1, Q_2, \ldots, Q_k$ are sets of nodes, we may use $\langle Q_1, Q_2, \ldots, Q_k \rangle$ to represent the set $\{\langle x_1, x_2, \ldots, x_k \rangle : x_i \in Q_i \text{ for } 1 \leq i \leq k\}$ of nodes. We may use $\langle\langle Q_1, \ldots, Q_k \rangle, \langle Q_1', \ldots, Q_k' \rangle, \langle Q_1'', \ldots, Q_k'' \rangle\rangle$ to refer to a set of vector m-resolutions in a similar way. Even such a representation might become cumbersome if k is larger than 3 or 4.

We now show how the vector m-resolution proof generated by "match" can be used to guide the search for a proof in the original space. Suppose $f_1, f_2, \ldots, f_k$ are m-abstraction mappings. Let $R(\bar{x}, C)$ be the following relation: $x_1 \in f_1(C) \wedge x_2 \in f_2(C) \wedge \ldots \wedge x_k \in f_k(C)$. Extend R to a relation between proofs in the usual way. Then it is easy to show that if T is an m-resolution proof, there exists a vector m-resolution proof W such that R(W, T) is true. Such a proof will satisfy $\Pi_i(W, T_i)$ for $1 \leq i \leq m$, for some T_i such that $T \underset{f_i}{\rightarrow} T_i$ is true.

Note that if T is a proof from S, and if R(W, T) is true, and if $\Pi_i(W, T_i)$ is true for $1 \leq i \leq k$, then $(R1; R2)(T_1, T_2, \ldots, T_k)$ will be true, where R1 and R2 are defined as follows:

$R1(D_1, D_2, \ldots, D_k)$ is true iff $(\exists C \in S)(D_i \in f_i(C), 1 \leq i \leq k)$.

$R2(D_1, D_2, \ldots, D_k)$ is true iff $(\exists C)(D_i \in f_i(C), 1 \leq i \leq k)$.

Suppose we are looking for a proof of m-clause C' from S. Let D_i' be arbitrary m-clauses such that $D_i' \in f_i(C')$, $1 \leq i \leq k$. Let V_i be an m-resolution proof from $f_i(S)$ such that V_i contains isomorphic copies of all minimal depth d m-resolution proofs of D_i' from $f_i(S)$, for $1 \leq i \leq k$. If there is a proof T of C from S such that T has depth d, then there are proofs T_i of D_i' from $f_i(S)$, for $1 \leq i \leq k$, such that $T \xrightarrow[f_i]{} T_i$, and such that T_i is an initial sub-proof of V_i. Therefore $(R1; R2)(T_1, \ldots, T_k)$ will be true. Therefore "match(V_1, V_2, ..., V_k, V, R1, R2)" will generate some proof W such that $\Pi_i(W, T_i)$ is true for $1 \leq i \leq k$. To be precise, W will be an initial sub-proof of the proof generated by "match". Note that R(W, T) will also be true. Therefore a reasonable search strategy is to call "match" to generate such a W, and to use "ndfindm" on W to obtain T. To do this, it is necessary to modify "ndfindm" to handle vector m-resolution proofs. This gives a theorem proving strategy "proofsearch4" which is entirely analogous to "proofsearch2" except that "proofsearch4" uses more than one m-abstraction at the same time. We now describe "proofsearch4". The procedure "proofsearch4" looks for proofs in order of increasing depth. The procedure uses a non-deterministic search analogous to "ndfindm" but for <u>vectors</u> of multiclauses.

```
procedure proofsearch4 (S, C', f_1, f_2, ..., f_k, D_1', D_2', ..., D_k');
  [look for a proof of C' from S, using the not necessarily distinct
   m-abstraction mappings f_1, f_2, ..., f_k and m-clauses D_i' such that
   D_i' ε f_i(C') for 1 ≤ i ≤ k.  This is a complete strategy.]
  let M1(D_1, D_2, ..., D_k) be the relation on m-clauses
    (∃C ε S)(D_i ε f_i(C) for 1 ≤ i ≤ k);
  let M2(D_1, ..., D_k) be some relation on m-clauses such that
    (∃C)(D_i ε f_i(C) for 1 ≤ i ≤ k) implies M2(D_1, D_2, ..., D_k);
  for d = 1 to ∞ until C' is derived from S do
    [look for a proof of depth d]
    for i = 1 to k do
      V_i ← an m-resolution proof containing as initial sub-proofs
            isomorphic copies of all minimal proofs T of D_i' from
            f_i(S) such that T has depth d;
      [generate V_i as in "proofsearch2"]
    od;
    match(V_1, V_2, ..., V_k, V, M1, M2);
    for all N ε Nodes(V) such that N is initial in V do
      m-clauses(N) ← {C ε S: D_i ε f_i(C), 1 ≤ i ≤ k}
                     where label(N) = ⟨D_1, D_2, ..., D_k⟩ od;
    for all N ε Nodes(V) such that N is non-initial in V do
      m-clauses(N) ← ∅ od;
    loop
      while (C' has not been derived from S and there exists a vector
            m-resolution ⟨N1, N2, N3⟩ in VMRes(V) and m-clauses
            C1, C2, C3 such that
            a) C1 ε m-clauses(N1) and C2 ε m-clauses(N2)
            b) C3 is an m-resolvent of C1 and C2
            c) C3 ∉ m-clauses(N3)
            d) for all i, 1 ≤ i ≤ k, D_i ε f_i(C3) where ⟨D_1, D_2, ..., D_k⟩ =
            label(N3));
```

```
      add C3 to m-clauses(N3);
    repeat
  od
end proofsearch4;
```

As with "proofsearch1", "proofsearch2", and "proofsearch3", if the abstracted spaces are all generated exhaustively, we can restrict the m-resolutions from S according to any complete theorem proving strategy and still obtain "proofsearch4" as a complete theorem proving strategy. Furthermore, if the m-abstractions satisfy suitable properties as indicated earlier, we can even restrict the abstracted search according to a complete theorem proving strategy. These combinations of m-abstraction and complete strategies restrict the possible m-resolutions while maintaining a "global" theorem proving strategy. That is, the choice of which m-resolutions to do is influenced in a non-trivial way by the structure of the problem as a whole, rather than by which clauses can resolve according to certain criteria.

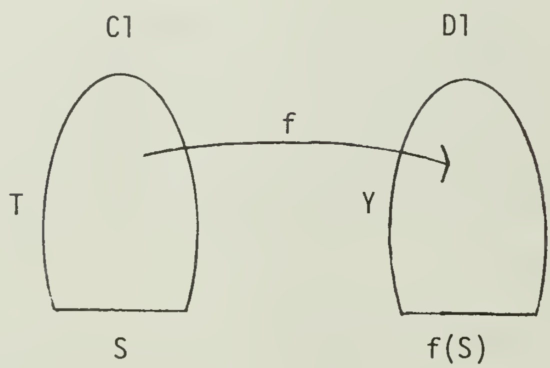
If instead of looking for a proof of a particular m-clause C' from a set S of m-clauses, we are looking for a proof from S of any one of a set S1 of m-clauses, the procedure "proofsearch2" can be modified to do this. Let f be an m-abstraction mapping. Suppose we are looking for a proof at depth d; the general idea is to generate all minimal proofs T from f(S) such that T has depth d and such that $(\exists C \in S1)(\mathrm{Result}(T) \in f(C))$. In order to do this, we need to be able to test for an m-clause D whether there exists $C \in S1$ such that $D \in f(C)$. When all such proofs T have been generated, they can be used to guide the search for a proof from S.

If f and S1 satisfy certain properties, then more efficient methods exist, and "proofsearch3" and "proofsearch4" can be used. Suppose for example that m-abstraction mapping f is defined in terms of a set F of literal mappings. Suppose S is a set of ordinary clauses. We are given an ordinary clause C' and want to determine whether there is an m-resolution proof from S of an m-clause C1 such that Set(C1) = C'. (For this derivation of C1, we consider the clauses in S to be m-clauses.) By Theorem 2.2, such an m-resolution proof exists if C' is derivable from S by ordinary resolution.

Suppose D ε f(C), where we consider C' as an m-clause for this purpose. Thus there exists a literal mapping $\emptyset$ ε F such that D = $\emptyset$(C). Now, if Set(C1) = C, it follows that Set($\emptyset$(C1)) = Set(D). Therefore, if C1 is any m-clause such that Set(C1) = C', then there exists D1 ε f(C1) such that Set(D1) = Set(D).

Suppose T is a minimal m-resolution proof from S of some m-clause C1 such that Set(C1) = C'. Then there is an m-resolution proof Y from f(S) such that $T \underset{f}{\rightarrow} Y$ and such that Result(Y) = D1. See figure 2. In order to search for such proofs T, then, it suffices to generate all proofs Y such that Set(Result(Y)) = Set(D) and use them to guide the search for T. In this way, "proofsearch2" can be adapted to search for m-resolution proofs of any m-clause C1 such that Set(C1) = C'. Moreover, since such a proof Y exists for <u>all</u> m-abstractions D of f(C'), we can use more than one m-abstraction of C' at the same time in the search for T. In this way, "proofsearch3" and "proofsearch4" can be used for this problem. Even if some m-clause C1 such that Set(C1) = C' is

derived from S by m-resolution, it does not follow that C' can be derived from S by ordinary resolution. All we know is that some clause <u>subsuming</u> C' can be derived from S by ordinary resolution, by Theorem 2.4.



M-clauses and ordinary clauses

Figure 2

This idea can be extended further. Suppose we are given a set S of ordinary clauses and an ordinary clause C'. We want to determine if C' is a logical consequence of S by using m-abstraction search techniques. Now, C' is a logical consequence of S <u>iff</u> there is a clause C" derivable from S by ordinary resolution such that C" subsumes C' [2]. Also, if f is an m-abstraction mapping defined in terms of literal mappings, then by Theorem 2.7 it follows that for every D' ε f(C'), there exists D" ε f(C") such that D" subsumes D'. If there is an ordinary resolution proof of C" from S, then there will be a minimal m-resolution proof T of C1 from S for some C1 such that Set(C1) = C".

By reasoning as before, there will be an abstraction D1 of C1 such that Set(D1) subsumes Set(D'). Also, there will be an m-resolution proof Y from f(S) such that $T \underset{f}{\rightarrow} Y$ and such that D1 = Result(Y). Therefore, to search for such a proof T, it suffices to search for proofs Y from f(S) such that Set(Result(Y)) subsumes Set(D'), and use these proofs to guide the search for T. In this way, "proofsearch2" can be adapted to test if an ordinary clause C' is a logical consequence of a set S of ordinary clauses. Since such proofs Y exist for <u>all</u> D' ε f(C') if C' is a logical consequence of S, the procedures "proofsearch3" and "proofsearch4" can also be so adapted. Note that if some m-resolution proof T from S of some m-clause C1 such that Set(C1) subsumes C' is found, then we know by Theorem 2.4 that some ordinary clause C2 such that C2 subsumes Set(C1)(hence C') is derivable from S by ordinary resolution. Thus C' is a logical consequence of S iff such a proof T exists.

5. BOUNDED M-CLAUSES

One disadvantage of m-clauses is that there are so many of them. The set of ordinary clauses over k distinct predicate symbols is finite, but the set of m-clauses over k distinct predicate symbols is infinite. This could result in a larger search space for the various abstraction-based theorem proving strategies. We now show how to overcome this problem to some degree, while retaining the advantages of m-abstractions. The general idea is to keep less information about how many occurrences of a literal there are in an m-clause. For example, we may specify that a certain literal occurs at least twice in an m-clause.

Definition: A bounded multiset is a multiset in which the multiplicities of the elements may be 0, 1, 2, ..., b-1, or ∞ for some bound b. A multiplicity of ∞ signifies that the element occurs at least b times. We call b the bound of the multiset. For practical purposes, an element with bound ∞ is considered to occur infinitely many times in the multiset. We only consider bounds b such that $b \geq 1$.

Definition: If A is a bounded multiset, then Set(A) is the set $\{x: \text{mult}(x, A) > 0\}$.

We define bounded addition $+^b$ and bounded subtraction $-^b$ of bounded integers as follows:

$x +^b \infty = \infty +^b x = \infty$ for all x

$x +^b y = x + y$ if $x \neq \infty$, $y \neq \infty$ and $x + y < b$

$x +^b y = \infty$ if $x \neq \infty$, $y \neq \infty$ and $x + y \geq b$

$x -^b y = x - y$ if $x \neq \infty$ and $y \leq x$

$x -^b y = 0$ if $x \neq \infty$ and($y = \infty$ or $y \geq x$)

$\infty -^b 0 = \infty$

$\infty -^b x = \{\infty, b-1, b-2, \ldots, b-x\}$ if $x \neq \infty$, $x \neq 0$

$\infty -^b \infty = \{\infty, b-1, b-2, \ldots, 2, 1, 0\}$

The meaning of the sets is that the operation can yield more than one possible outcome. Thus $\infty -^b x$ can have any value between $b - x$ and ∞ if $x \neq \infty$ and $x \neq 0$. These definitions are obtained as follows: Let $\emptyset_b(x)$ be ∞ if $x \geq b$ and $\emptyset_b(x) = x$ if $0 \leq x < b$. Then if $x + y = z$ for ordinary nonnegative integers x, y, and z, we say $\emptyset_b(x) +^b \emptyset_b(y) = \emptyset_b(z)$. If $x \dot{-} y = z$ for ordinary nonegative integers x, y, and z, we say $\emptyset_b(x) -^b \emptyset_b(y) = \emptyset_b(z)$. Here $x - y$ is defined to be $\max(0, x-y)$.

<u>Definition</u>: If A and B are bounded multisets with bound b, then $A \cup B$ and $A - B$ are defined as follows:

$\text{mult}(x, A \cup B) = \text{mult}(x, A) +^b \text{mult}(x, B)$

$\text{mult}(x, A - B) = \text{mult}(x, A) -^b \text{mult}(x, B)$

Note that $\text{Set}(A \cup B) = \text{Set}(A) \cup \text{Set}(B)$ and $\text{Set}(A) - \text{Set}(B) \subseteq \text{Set}(A - B)$.

<u>Example</u>: Suppose $b = 2$. Suppose A is $\{\infty*P, 1*Q\}$ and B is $\{1*P, 1*Q, 1*R\}$. Then

$A \cup B = \{\infty*P, \infty*Q, 1*R\}$

$A - B = \{\infty*P\}$ or $\{1*P\}$

$B - A = \{1*R\}$.

Given an ordinary multiset C, let $\emptyset_b(C)$ be defined by $\text{mult}(x, \emptyset_b(C)) = \emptyset_b(\text{mult}(x, C))$. Thus $\emptyset_b(C)$ is a bounded multiset, with bound b.

The bounded multiset operations are defined so that if C1 and C2 are ordinary multisets, then $\emptyset_b(C1) \cup \emptyset_b(C2) = \emptyset_b(C1 \uplus C2)$ and $\emptyset_b(C1) - \emptyset_b(C2) = \emptyset_b(C1 - C2)$. Note as before that set difference is a "nondeterministic" operation.

<u>Definition</u>: If A is a bounded multiset and f is a function on elements of A, then f(A) is defined by $\mathrm{mult}(y, f(A)) = \sum_{f(x)=y} \mathrm{mult}(x, A)$ where <u>bounded</u> addition is used for the sum. In particular, if A is a bounded multiset of literals and α is a substitution, then $A\alpha$ is defined in this way. Thus for bounded multisets with bound 2, $\{1{*}P(z), 1{*}P(a), 1{*}Q(z)\}\{z \leftarrow a\} = \{\infty{*}P(a), 1{*}Q(a)\}$. Here $\{z \leftarrow a\}$ is the substitution replacing z by a.

<u>Definition</u>: A <u>bounded</u> m-clause is a bounded multiset of literals.

5.1 *Bounded M-Resolution*

<u>Definition</u>: Suppose C1 and C2 are bounded m-clauses. Suppose $A1 \subseteq C1$ and $A2 \subseteq C2$. Suppose $\alpha 1$ and $\alpha 2$ are most general substitutions such that there exists a literal L such that $\mathrm{Set}(A1\alpha 1) = \{L\}$ and $\mathrm{Set}(A2\alpha 2) = \{\bar{L}\}$. Then $(C1 - A1)\alpha 1 \cup (C2 - A2)\alpha 2$ is a bounded m-resolvent of C1 and C2.

Note that ordinary clauses can be viewed as bounded m-clauses, although the resolution operation is different.

One reason for the usefulness of bounded m-resolution is that it "lifts" to ordinary m-resolution. That is, B3 is a bounded m-resolvent of B1 and B2 iff there exist ordinary m-clauses C1, C2, and C3

such that C3 is an m-resolvent of C1 and C2, and such that $\emptyset_b(C1) = B1$, $\emptyset_b(C2) = B2$, and $\emptyset_b(C3) = B3$. Here b is the bound as usual.

Examples: Suppose $B1 = \{1{*}P, 1{*}Q\}$ and $B2 = \{\infty{*}\bar{P}\}$ are bounded m-clauses with bound 2. Their bounded m-resolvents are the following clauses:

$\{\infty{*}\bar{P}, 1{*}Q\}$

$\{1{*}\bar{P}, 1{*}Q\}$

Suppose $B1 = \{1{*}P, 1{*}Q\}$ and $B2 = \{1{*}\bar{P}, 1{*}Q\}$, with bound 2 as before. Then the only bounded m-resolvent is $\{\infty{*}Q\}$. Suppose $B1 = \{\infty{*}P\}$ and $B2 = \{\infty{*}\bar{P}, 1{*}Q\}$, with $b = 2$. Then the bounded m-resolvents are the following clauses:

$\{\infty{*}P, \infty{*}\bar{P}, 1{*}Q\}$

$\{\infty{*}P, 1{*}\bar{P}, 1{*}Q\}$

$\{\infty{*}P, 1{*}Q\}$

$\{1{*}P, \infty{*}\bar{P}, 1{*}Q\}$

$\{1{*}P, 1{*}\bar{P}, 1{*}Q\}$

$\{1{*}P, 1{*}Q\}$

$\{\infty{*}\bar{P}, 1{*}Q\}$

$\{1{*}\bar{P}, 1{*}Q\}$

$\{1{*}Q\}$

Theorem 5.1: Suppose S is a set of multiclauses and C' is derivable from S by m-resolution. Let $\emptyset_b(S)$ be $\{\emptyset_b(C) : C \in S\}$. Thus $\emptyset_b(S)$ is a set of bounded multiclauses with bound b. Then $\emptyset_b(C')$ is derivable from $\emptyset_b(S)$ by bounded m-resolution.

Corollary: Suppose S is a set of ordinary clauses and clause C is derivable from S by ordinary resolution. Then there is a clause C' derivable from S using bounded m-resolution such that Set(C') = C. (Recall that the bound is greater than or equal to one.)

Thus m-resolution proofs and ordinary resolution proofs can be transformed into bounded m-resolution proofs. In fact, this can be done so that the bounded m-resolution proof always has the same shape as the original one. In addition, we can show that a set S of clauses is inconsistent iff NIL (the empty clause) is derivable from S by bounded m-resolution. For this, we view ordinary clauses as bounded multiclauses with each literal having multiplicity one (or ∞ if b = 1). Furthermore, if C' is derivable from set S of clauses by bounded m-resolution, then there is a clause C derivable from S by ordinary resolution such that C subsumes Set(C').

5.2 *Bounded M-Abstractions*

Definition: A bounded m-abstraction mapping is a function f mapping ordinary multiclauses into sets of bounded multiclauses, satisfying the following properties:

1. If C3 is an m-resolvent of C1 and C2, and D3 ε f(C3), then there exist D1 ε f(C1) and D2 ε f(C2) such that D3 is a bounded m-resolvent of D1 and D2.
2. f(NIL) = {NIL}.

Note that D1, D2, D3 are bounded multiclauses and C1, C2, and C3 are ordinary multiclauses. Also, note that the function $\emptyset_b$ is itself a bounded m-abstraction mapping with bound b. In addition, if f

is an ordinary m-abstraction mapping then $\emptyset_b \circ f$ is a bounded m-abstraction mapping. In this way, we can get bounded m-abstraction mappings from all of the ordinary m-abstraction mappings described earlier. We could define abstraction mappings from bounded multiclauses to bounded multiclauses in the same way. Also, we could prove the appropriate theorems about the closure properties of bounded m-abstraction mappings under union and composition.

All the search strategies for m-abstraction mappings can be applied to bounded m-abstraction mappings as well. For example, more than one bounded m-abstraction mapping can be used together in the search for a proof. Also, bounded m-abstractions can be used to test if a clause is a logical consequence of a set of clauses. Certain bounded m-abstractions are particularly useful. For example, if f is the propositional m-abstraction or a semantic m-abstraction with a finite domain, then $\emptyset_b \circ f$ is a bounded m-abstraction mapping with a finite range. That is, $\{D: (\exists C)\ D = (\emptyset_b \circ f)(C)\}$ is finite. Hence we can exhaustively list this set, as well as the set $\{\langle D1, D2, D3\rangle : D3$ is a bounded m-resolvent of D1 and D2$\}$. The various search strategies can use this information without having to recompute it for each depth. In addition, by appropriate hash coding or indexing schemes, this information can be compactly stored and efficiently retrieved. In this way, we get many of the benefits of multiclauses with the additional benefit of a finite abstracted space. Of course, the search strategy is less restrictive than with m-clauses, but this seems more than compensated for by the finiteness of the abstracted space. Increasing the bound will yield

a more restrictive search strategy, at the expense of increasing the size of the abstracted space. A bound near 2 would seem best for most applications. A bound of 1 is really too small, since this bound does not distinguish between one occurrence of a literal (the usual case) and more than one occurrence of a literal.

6. INTERVAL MULTISETS

We now define a concept of multisets more general than bounded multisets and ordinary multisets. In an interval multiset, we have a partition P of the integers (usually into intervals) and only keep information about which block of P the multiplicities occur in. In this way, we can distinguish "many" occurrences of a literal from "few" occurrences without having to have a large number of multiplicities.

Definition: Suppose P is a partition of the non-negative integers. Thus P is a set of disjoint "blocks" $\{I_1, I_2, \ldots\}$ whose union is the non-negative integers. Then a bounded multiset is a set A together with a multiplicity mult(x, A) for each x in A. Also, mult(x, A) is a block of P for all x. We will assume that {0} is always one of the blocks of P. This seems to be a reasonable restriction.

Definition: Suppose I_1 and I_2 are sets of integers. Then

$$I_1 + I_2 = \{x + y\colon x \in I_1, y \in I_2\}$$

$$I_1 \dot{-} I_2 = \{x \dot{-} y\colon x \in I_1, y \in I_2\}$$

Here $x \dot{-} y$ is 0 if $x \leq y$, $x - y$ otherwise.

Definition: If A and B are interval multisets, then $C = A \cup B$ if for all x, $\mathrm{mult}(x, C) \cap [\mathrm{mult}(x, A) + \mathrm{mult}(x, B)] \neq \emptyset$. Thus mult(x, C) is some block of P having non-empty intersection with mult(x, A) + mult(x, B). Similarly, $C = A - B$ if $\mathrm{mult}(x, C) \cap [\mathrm{mult}(x, A) \dot{-} \mathrm{mult}(x, B)] \neq \emptyset$ for all x. Thus $A \cup B$ and $A - B$ are "nondeterministically defined"; they can have more than one value. Hence it is not strictly rigorous to write $C = A \cup B$ or $C = A - B$.

Definition: If A is an interval multiset and f is a mapping from elements of A, then $f(A)$ is defined so that for all y, $\text{mult}(y, f(A)) \cap (\sum_{f(x)=y} \text{mult}(x, A)) \neq \emptyset$. Thus if $\{x_1, x_2, \ldots, x_k\}$ is $\{x: f(x) = y\}$ and $I_1, I_2, \ldots, I_k$ are the multiplicities of $x_1, \ldots, x_k$, respectively, then $\text{mult}(y, f(A))$ must have non-empty intersection with $I_1 + I_2 + \ldots + I_k$. (We are assuming $x_1, x_2, \ldots, x_k$ are all distinct.) Note that $f(A)$ is also defined "nondeterministically".

Definition: An interval m-clause is an interval multiset whose elements are literals.

Definition: If A is an interval multiset, then $\text{Set}(A) = \{x: \text{mult}(x, A) \neq \{0\}\}$.

Definition: Suppose P is a partition of the non-negative integers. Then the mapping ψ_p from ordinary multiclauses to interval multiclauses is defined as follows:

For all x, $\text{mult}(x, C) \in \text{mult}(x, \psi_p(C))$. In other words, the multiplicity of x in $\psi_p(C)$ must be a block of P containing the multiplicity of x in C. Here C is an ordinary multiclause.

Definition: Suppose C1 and C2 are interval multiclauses with partition P. Then we say C3 is an interval m-resolvent of C1 and C2 if there exist ordinary multiclauses B1, B2, and B3 such that B3 is an m-resolvent of B1 and B2, and such that $\psi_p(B1) = C1$, $\psi_p(B2) = C2$, and $\psi_p(B3) = C3$. We could also define interval m-resolvents as $(C1 - A1)\alpha 1 \cup (C2 - A2)\alpha 2$ but we choose the above approach for simplicity.

It is easy to verify that if C' is derivable from set S of m-clauses by m-resolution, then $\psi_p(C')$ is derivable from set $\psi_p(S)$ of

interval m-clauses by interval m-resolution. Hence if S is inconsistent, NIL (i.e., ψ_p(NIL)) is derivable from $\psi_p(S)$ by interval m-resolution. The converse is also true, because {0} is one of the blocks of P.

We could define interval m-abstractions in the usual way and prove the relevant theorems about closure of interval m-abstractions under union and composition. It appears that interval m-abstractions will be less useful than bounded m-abstractions, since we would not expect the number of occurrences of a literal in a multiclause to get very large. However, there may be applications in which large numbers of the same literal do occur in a multiclause.

7. PARTITIONED SEMANTIC ABSTRACTIONS

Just as we obtained interval m-clauses from ordinary m-clauses by partitioning the multiplicities, so we can obtain new abstractions from semantic abstractions by partitioning the domain of the abstraction. The resulting proof technique resembles human use of reasoning with diagrams. In particular, partitioned semantic abstractions correspond to incompletely specified diagrams, the kind one may draw on the blackboard with dots or scribbles to indicate unspecified parts of the diagram. This seems to correspond to the kind of reasoning process that humans (at least the author) use to prove real theorems.

Recall that a semantic abstraction maps clauses onto clauses of the form $\{L_1, \ldots, L_k\}$ where each L_i is of the form $P(a_1, \ldots, a_n)$ or $\neg P(a_1, \ldots, a_n)$ and a_i are elements of the domain $\mathcal{D}$ of the interpretation. Suppose P is a partition of $\mathcal{D}$. With the literal L we associate the

literal $f_P(L)$ defined as follows: If L is of the form $P(a_1, \ldots, a_n)$ then $f_P(L)$ is $P(A_1, \ldots, A_n)$, where $a_i \varepsilon A_i$ and A_i are blocks of the partition P. If L is of the form $\neg P(a_1, \ldots, a_n)$ then $f_P(L)$ is $\neg P(A_1, \ldots, A_n)$, with notation as above. Finally, with the clause $C = \{L_1, \ldots, L_k\}$ we associate the clause $f_P(C) = \{L_1', \ldots, L_k'\}$ where L_i' is $f_P(L_i)$ for $1 \leq i \leq k$.

It is not difficult to show using Theorem 2.2 of Part I that if g is a semantic abstraction with domain $\mathcal{D}$, then $f_P \circ g$ is also an abstraction. However, $f_P \circ g$ may be <u>finite</u> (that is, $\{f_P \circ g(C)\}$ may be finite) even if g is not. In this way, finite abstractions can be obtained in a fairly natural way from semantic abstractions. We call $f_P \circ g$ a <u>partitioned semantic abstraction</u>. We do not remember individual elements of the domain, but only which block of P they belong to. For example, we may remember only the congruence class of an integer modulo a prime. Or we may divide integers into "big" integers and "little" integers for reasoning about inequalities. Thus we get abstractions that correspond to incompletely specified diagrams. The significance of this result is not that we have a formalism for incompletely specified diagrams, but that the formalism is quite general and leads to a general theorem proving strategy. We can have partitioned semantic m-abstractions, in the usual way. Partitioned semantic bounded m-abstractions are also possibilities.

8. OTHER ABSTRACTIONS

Suppose f is the ground abstraction. Let $\emptyset$ be any literal mapping

such that $\emptyset(\bar{L}) = \overline{\emptyset(L)}$. Let g be the abstraction defined by g(C) = $\{\emptyset(C)\}$ for ground clauses C. (Here $\emptyset(C) = \{\emptyset(L): L \in C\}$ as usual.) Then by Theorem 2.2 of Part I, $g \circ f$ is also an abstraction. In this way, we can show that semantic abstractions and partitioned semantic abstractions are abstractions. Suppose L is the ground literal $P(t_1, \ldots, t_n)$. To obtain semantic abstractions, we define $\emptyset(L)$ to be $P(a_1, \ldots, a_n)$, where a_i is the value of t_i in the given interpretation I. Also, we define $\emptyset$ so that $\emptyset(\bar{L}) = \overline{\emptyset(L)}$. For partitioned semantic abstractions, $\emptyset(L) = P(A_1, \ldots, A_n)$ where $a_i \in A_i$ and A_i is a block of the given partition of the domain of I. Also, $\emptyset(\bar{L}) = \overline{\emptyset(L)}$ as usual. We can extend this idea further.

Suppose I_1 and I_2 are two interpretations, with domains $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Let L be the ground literal $P(t_1, \ldots, t_n)$, as above. Let a_i and b_i be the values of t_i in I_1 and I_2, respectively. Let p be a new function symbol (representing "pairing"). Define $\emptyset$ by $\emptyset(L) = P(p(a_1, b_1), \ldots, p(a_n, b_n))$, and $\emptyset(\bar{L}) = \overline{\emptyset(L)}$. This yields another abstraction, which is the "product" of two semantic abstractions, in a sense. We can also define $\emptyset(L) = P(p(A_1, B_1), \ldots, p(A_n, B_n))$ where the A_i are blocks of some partition of $\mathcal{D}_1$ and the B_i are blocks of some partition of $\mathcal{D}_2$ and $a_i \in A_i$, $b_i \in B_i$ for $1 \le i \le n$. We define $\emptyset(\bar{L})$ to be $\overline{\emptyset(L)}$ as usual. Many more abstractions can be obtained in this way. For example, we can take the "product" of any number of semantic abstractions. In a similar way, we can obtain m-abstractions and bounded m-abstractions based on such literal mappings $\emptyset$. Thus we can take the product of two semantic m-abstractions, and so on. It is clear that there are a great many possible ways in which abstractions and related concepts can be used to obtain complete theorem proving strategies.

9. CONCLUSIONS

The concepts of abstraction, m-abstraction, and bounded m-abstraction lead to a wide variety of new, complete uniform proof procedures for the first order predicate calculus. The same strategies probably apply to higher order logics with slight modification. These strategies all make use of a simplified proof from a simplified set of clauses (m-clauses, bounded m-clauses) to guide the search for a proof from the original set of clauses (m-clauses, bounded m-clauses). This is a much more "global" technique than current uniform proof procedures use. That is, each inference is controlled in a more meaningful way by the structure of the problem as a whole, rather than by properties local to the clauses involved in the inference. Also, near the end of the search, the abstracted clauses are more restricted than in the middle because an abstraction of the "goal clause" must be derivable from them in fewer steps. Thus the search space tends to get small as the depth of inference increases towards its maximum value. Furthermore, these methods permit depth-first search and subgoaling more naturally than most resolution strategies do. In fact, we are working on other methods which use abstractions together with backward reasoning, and which rely more heavily on semantics to decide which subgoals are achievable.

The abstractions based on particular interpretations seem to be especially interesting, because they come close to formalizing the idea of proving a theorem for a particular example, a technique frequently used by humans. Abstractions corresponding to interpretations with a finite domain are promising, because they lead to a finite abstracted search space when used with bounded m-clauses.

Strategies based on "multiclauses" and abstraction turn out to be simpler and more elegant than strategies based on ordinary clauses and abstraction. (A multiclause is a multiset of literals). These multiclause strategies permit the use of several abstractions at the same time in a natural way. The combination of multiclauses and abstraction seems to be a significant new development. The use of more than one level of abstraction is another promising possibility which we do not explore.

Structured programs are given to illustrate some of the strategies presented. Experience with implementations of these programs is necessary to determine the practical value of the techniques presented here. However, due to their basic underlying simplicity and elegance, these strategies should be relatively easy and straightforward to implement.

More work remains to be done in extending the concept of abstraction to other systems of inference rules and to higher order logics. Can abstraction be applied to automatic program generation, for example? Perhaps abstraction could lead to fast theorem provers even for the propositional calculus. Also, it would be desirable to combine abstraction with a more meaningful use of semantics and with a strategy for equality. The compatibility of abstraction with conventional strategies such as locking resolution can also be explored. Finally, we plan to investigate combinations of abstraction with backward reasoning from a goal. The reason for hoping that abstraction will lead to better theorem provers is that it seems to be qualitatively different from the kinds of theorem proving strategies considered in the past, in restricting the search by use of global information about the problem to be solved.

The generality of this approach is also attractive, as well as the ability to use specialized knowledge concerning which abstractions are helpful for which problem domains.

10. REFERENCES

1. Robinson, J. A., Automatic deduction with hyper-resolution, Internat. J. Comput. Math 1 (1965) 227-234.

2. Kowalski, R., The case for using equality axioms in automatic demonstration, Symp. Automatic Demonstration (Springer-Verlag, New York, 1970) 112-127.

BIBLIOGRAPHIC DATA SHEET

1. Report No. UIUCDCS-R-79-965

2.

3. Recipient's Accession No.

4. Title and Subtitle

Theorem Proving with Abstraction, Part II

5. Report Date

March 1979

6.

7. Author(s)

David A. Plaisted

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

NSF MCS 77-22830

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D. C.

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

The concept of an abstraction was defined in Part I of this paper, and some theorem proving strategies based on abstraction were presented. The basic idea is to use the solution of a simple "abstracted" problem as a guide to the solution of a more complicated problem. This idea was formalized to yield a wide class of complete theorem proving strategies for the first-order predicate calculus. In part II, m-abstractions are defined, and their advantages are discussed. They operate on "multiclauses", which are multisets of literals. Several elegant strategies based on m-abstractions are presented. Next, bounded multiclauses are introduced, together with abstractions on them. These have most of the advantages of ordinary multiclauses, but restrict the size of the abstracted search space more. All strategies considered in Part II are complete. Finally, some new classes of abstraction and m-abstraction mappings are presented.

17. Key Words and Document Analysis. 17a. Descriptors

Theorem proving, abstraction, analogy, first-order predicate calculus, resolution, multisets.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement

19. Security Class (This Report) UNCLASSIFIED

20. Security Class (This Page) UNCLASSIFIED

21. No. of Pages

22. Price